|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | https://lh4.googleusercontent.com/proxy/YA9Xoqs7jhpeuwrEjwhdi_EVSCDwUdpr72V-2YHZ2lz2y1FaqityK8c8RlZRTvUDEw3Y2TekyGNi07wcREil5Ez3ii80dA-DE8G6HAQjEmJVz8W32Wy2uaDAWwuZs6uPZtJp2zrUJ_Qps2T1CUmSpuPR8dk2XA=w128-h144-k-no | | | **Document Ref.:** | | |  | |
| **Version No.:** | | |  | |
| **Date:** | | |  | |
| **Copy No.:** | | |  | |
| **Project Name:** |  | | | | | | |
| **Project Code:** |  | | | | | | |
| **Status:** | **Draft** / ~~Current~~ / ~~Superceded~~ | | | | | | |
| **Document Type:** | ~~Controlled~~ / **Uncontrolled** | | | | | | |
| **<Title>** | | | | | | | |
| **<Summary>** | | | | | | | |
| **Prepared By:** | | | | **Reviewed By:** | | | |
| **Name** | | **Date** | | **Name** | | | **Date** |
|  | |  | |  | | |  |
|  | | |  |
|  | |  | | **Approved By:** | | | |
| **Name** | | | **Date** |
|  | |  | |  | | |  |
|  | | |  |
| **Distribution List** | | | | | | | |
| **Project Representative(s)** | | | | | **PESU Representative(s)** | | |
| 1. <Name1> 2. <Name2> 3. <Name3> | | | | | 1. <Name4> 2. <Name5> 3. <Name6> | | |

**TABLE OF CONTENTS**

[**Definitions, Acronyms and Abbreviations 2**](#_heading=h.1pxezwc)

[**References 2**](#_heading=h.49x2ik5)

[**Change History 2**](#_heading=h.tyjcwt)

[**1.0**](#_heading=h.2p2csry) **Introduction 2**

[**1.1**](#_heading=h.147n2zr) **Overview 2**

[**1.2**](#_heading=h.3o7alnk) **Purpose 2**

[**1.3**](#_heading=h.23ckvvd) **Scope 2**

[**2.0**](#_heading=h.ihv636) **Design Constraints, Assumptions and Dependencies 2**

[**3.0**](#_heading=h.32hioqz) **Design Description 2**

[**3.1**](#_heading=h.1hmsyys) **<Module Name> 2**

[*3.1.1*](#_heading=h.41mghml) *Class Name 1 2*

[**3.1.1.1**](#_heading=h.1ksv4uv) **Class Description 1 2**

[**3.1.1.2**](#_heading=h.2grqrue) **Data Members 1 2**

[3.1.1.2.1](#_heading=h.2jxsxqh) Method 1 2

[3.1.1.2.2](#_heading=h.z337ya) Method 2 2

[3.1.1.2.3](#_heading=h.3j2qqm3) Method n 2

[*3.1.2*](#_heading=h.vx1227) *Class Name 2 2*

[**3.1.2.1**](#_heading=h.4i7ojhp) **Class Description 2 2**

[**3.1.2.2**](#_heading=h.2xcytpi) **Data Members 2 2**

[3.1.2.2.1](#_heading=h.1ci93xb) Methods 1 2

[3.1.2.2.2](#_heading=h.3whwml4) Methods 2 2

[3.1.2.2.3](#_heading=h.2bn6wsx) Methods n 2

[*3.1.3*](#_heading=h.qsh70q) *Class Description n 2*

[**4.0**](#_heading=h.3fwokq0) **Traceability Matrix 2**

**Definitions, Acronyms and Abbreviations**

This section provides for definition of all terms, acronyms and abbreviations required for interpreting the Low Level Design Document. Well known abbreviations need not be stated.

**Change History**

This section describes the details of changes that have resulted in the current low-level Design document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Date** | **Document Version No.** | **Change Description** | **Reason for Change** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 1.0 **Introduction**

## ***Overview***

LLD has different modules, data description and class description for each module along with the functions to execute each module.

## ***Purpose***

## The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code. Low-level design is created based on the high-level design. LLD describes the class diagrams with the methods and relations between classes and program specs.

## ***Scope***

This project is implemented to help organizations to rate and rank automobiles over a variety of universally accepted testing parameters with the help of a robust and efficient web-based software, which is in turn used by the public to make informed decisions while purchasing these automobiles.

The software does not use real-time data and requires the database to be populated before the software is operational.

# **Design Constraints, Assumptions and Dependencies**

This section provides the list of constraints, assumptions and dependencies.

# **Design Description**

This section describes the design with respect to functional modules.

## Security/Login Module

* + 1. **Security Admin:**
       1. **Class Description:**

This is used to represent the security administrator of the organisation. This class inherits all the attributes and methods of the existing user class. The additional methods in this class are deleteUser, blockUser and modifyClearance

* + - * 1. **blockUser:**

Purpose: This method is used to block a specific user.

Input: UID

Output: None

Parameters: UID

Exceptions: None

Pseudo code:

blockUser(UID)

{

user = retrieve user details from database based on UID

if(department has authorised the block request)

{

user status = blocked

add updated user to the database

}

}

* + - * 1. **modifyClearance:**

Purpose: This method is used to modify the clearance level of the user

Input: UID, Clearance

Output: None

Parameters: UID, Clearance

Exceptions: None

Pseudo code:

modifyClearance(UID, Clearance)

{

user = retrieve user details from database based on UID

if(department approves the clearance)

{

clearance = Clearance

add updated user to database

}

}

* + - * 1. **deleteUser:**

Purpose: This method is used to delete a user’s records

Input: UID

Output: None

Parameters: UID

Exceptions: None

Pseudo code:

deleteUser(UID)

{

if(department approves user deletion)

{

delete user record from database based on UID

}

}

* + 1. **User**
       1. **Class Description:**

This is used to represent each user in the organisation. This class has attributes Name, UID, Age and DepartmentID. It is a generalisation of new user, existing user and security admin. It has associations with Department and Account classes.

* + - 1. **Data Members:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Data Name** | **Access Modifiers** | **Initial Value** | **Description** |
| String | DepartmentID | Private | - | Each user belongs to a department that has a unique DepartmentID |
| String | Name | Private | - | Each user has a unique name |
| Int | Age | Private | - | Each user has an age. |
| String | UID | Private | - | Each user has a unique UID |

* + 1. **New User**
       1. **Class Description:**

This is used to represent the new users in the organisation. This is a specialization of the user class. This class inherits all attributes from the user class. The method in this class is signup.

* + - * 1. **signup**

Purpose: This method is used to create a new account for new users.

Input: Username, Password, Department

Output: None

Parameters: Username, Password, DepartmentID, UID

Exceptions: Username and password aren’t according to specified formats.

Pseudo code:

signup(username, password, department)

{

if(username and password are valid)

{

if(department authorizes new user)

{

UID = generate new UID

dID = get DepartmentID from database based on department

user = create new record in the database with username, password, UID, and DepartmentID

}

}

}

* + 1. **Existing User**
       1. **Class Description:**

This is used to represent the existing users in the organisation. This is a specialization of the user class. It inherits all the attributes of the user class. The methods in this class are getDetails and modifyDetails

* + - * 1. **login:**

Purpose: This method is used by the existing users to login to their accounts.

Input: username, password

Output: None

Parameters: username, password

Exceptions: The username or password is in an incorrect format.

Pseudo code:

login(username, password)

{

if(username and password are valid)

{

fetch and load user’s dashboard

}

}

* + - * 1. **logout**

Purpose: This method is used by existing logged in users to logout

Input: None

Output: None

Parameters: None

Exceptions: None

Pseudo code:

logout()

{

fetch and load the login page

}

* + - * 1. **getDetails:**

Purpose: This method is used to retrieve the details of the user.

Input: None

Output: The details of the user i.e Name, Age and Department.

Parameters: UID

Exceptions: None

Pseudo code:

getDetails(UID)

{

name,age,department = retrieve user details from database based on UID

display(name, age, department)

}

* + - * 1. **modifyDetails:**

Purpose: This method is used to modify the details of the user i.e. name and age

Input: new name, new age

Output: None

Parameters: UID, new name, new age

Exceptions: The new name and age don’t follow the specified formats.

Pseudo code:

modifyDetails(UID, new name, new age)

{

if(name and age are valid)

{

user = retrieve user row from database

name = new name

age = new age

add updated user to database

}

}

* + 1. **Department**
       1. **Class Description:**

This is used to represent the various departments of the organisation to which each user belongs to. This class has attributes DepartmentID, Department Name and Number of employees. The methods in this class are getEmpCnt, authorizeUser and confirmUserChanges. It has associations with user and security admin classes.

* + - 1. **Data Members:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Data Name** | **Access Modifiers** | **Initial Value** | **Description** |
| String | DepartmentID | Private | - | Each department has a unique DepartmentID |
| String | Department Name | Private | - | Each department has a unique department name |
| Int | Number of Employees | Private | - | Each department maintains the number of employees it contains. |

* + - * 1. **getEmpCnt():**

Purpose: This method is used to retrieve the number of employees in the department.

Input: None

Output: The number of employees in the department

Parameters: DepartmentID

Exceptions: None

Pseudo code:

getEmpCnt(DepartmentID)

{

Number of employees = get department row from database based on DepartmentID

return Number of employees

}

* + - * 1. **authorizeUser():**

Purpose: This method is used to authorize the entry of a new user into the department.

Input: None

Output: Status

Parameters: UID

Exceptions: The UID already exists in the department

Pseudo code:

authorizeUser(UID)

{

if(UID not in department)

{

// if department has indeed hired a new employee

status=accepted

}

else

status=rejected

return status

}

* + - * 1. **confirmUserChanges():**

Purpose: This method is used to confirm the changes made by the security admin to a user’s account.

Input: None

Output: Status

Parameters: UID, changes

Exceptions: The department doesn’t have the user specified.

Pseudo code:

confirmUserChanges(UID, changes)

{

if department is ok with the changes

status = accepted

else

status = rejected

return status

}

* + 1. **Account:**
       1. **Class Description:**

This is used to represent the accounts held by each user of the organisation. This class has attributes username and password. The methods in this class are getAccountDetails, modifyAccountDetails and changePassword. It has associations with the user and security admin classes.

* + - 1. **Data Members:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | Data Name | Access Modifiers | Initial Value | Description |
| String | Username | Private | - | Each end user has a unique username. |
| String | Password | Private | - | Each user has a unique password that is associated with the username. |

* + - * 1. **getAccountDetails():**

Purpose: This method is used to retrieve the account details of the user i.e. the username and password.

Input: None

Output: This method returns the username and password of the user

Parameters: UID

Exceptions: None

Pseudo code:

getAccountDetails(UID):

{

username, password = retrieve username and password from database based on UID

display(username, password)

}

* + - * 1. **modifyAccountDetails():**

Purpose: This method is used to modify the account details of the user i.e name change

Input: The new name

Output: None

Parameters: New name, UID

Exceptions: The new name is not according to the format specified(32-character long alphanumeric/special character sequence).

Pseudo code:

modifyAccountDetails(new name, UID):

{

user = retrieve user row from database based on UID

username = new name

add updated user to database

}

* + - * 1. **changePassword():**

Purpose: This method is used to change the password of the user

Input: Current password,New password

Output: None

Parameters: Current Password, New password, UID

Exceptions: The new password is not according to the format specified(30-character long alphanumeric/special character sequence). The user enters incorrect current password.

Pseudo code:

changePassword(current password, new password, UID)

{

if(password==current password)

{

user = retrieve user row from database based on UID

password = new password

add updated user to database

}

}

## Predictor Module

### End User:

#### Class Description:

This is used to represent the end user who uses the predictor module to predict automobile ranks. The user has attributes username and password and a method checkPassword to check if the entered password matches with the username. It has an association with the predictor tool class.

#### Data Members:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | Data Name | Access Modifiers | Initial Value | Description |
| String | Username | Protected | - | Each end user has a unique username. |
| String | Password | Private | - | Each user has a unique password that is associated with the username. |

* + - * 1. checkPassword(username,password)

Purpose: This method is used to check if the password given by the end user matches his/her username.

Input: This method takes the username and password provided by the user.

Output: This method outputs 1 if the given password is correct for the username and 0 if it does not match.

Parameters: This method takes the username and password of the end user as parameters.

Exceptions: The user may provide invalid username or password which could be in a format not expected. In this case an exception has to be raised.

Pseudo-code:

checkPassword(username, password){

pass=Retrieve password in database corresponding to given username

if(strcmp(pass,password)==0)

{

//Given password is correct

return 1;

}

return 0;

}

### Prediction Administrator

#### Class Description:

This is used to represent the predictor tool administrator who can alter the type of formula or the parameters in the parameter list. This type of user has access to functions of the predictor tool class such as addParam()- to add a parameter, alterParam()- to alter the value of an existing parameter and delParam()- to delete an existing parameter. The prediction administrator can also change the type of formula based on necessity using changeFormula() method.

#### Data Members:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | Data Name | Access Modifiers | Initial Value | Description |
| String | Username | Protected | - | Each predictor tool administrator has a unique username. |
| String | Password | Private | - | Each administrator has a unique password that is associated with the username. |

* + - * 1. checkPass(username,password)

Purpose: This method is used to check if the password given by the predictor tool administrator matches his/her username.

Input: This method takes the username and password provided by the administrator.

Output: This method outputs 1 if the given password is correct for the username and 0 if it does not match.

Parameters: This method takes the username and password of the administrator as parameters.

Exceptions: The user may provide invalid username or password which could be in a format not expected. In this case an exception has to be raised.

Pseudo-code:

checkPass(username, password){

pass=Retrieve password in database corresponding to given username and check if the username has admin privileges.

if(strcmp(pass,password)==0)

{

//Given password is correct

return 1;

}

return 0;

}

### Predictor tool

**3.2.3.1 Class Description:**

This is used to represent the predictor tool which consists of the Parameter list and Formula used for rank prediction of automobiles. It consists of several operations such as predict(), displayranks() and so on, which are used by the end user to predict, display ranks, sort outputs based on filters and so on. There are also operations provided to add, alter and delete parameter values.

**3.2.3.2 Data Members:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Data Name** | **Access Modifiers** | **Initial Value** | **Description** |
| User defined | Parameter list | Protected | - | This is the list of parameters with their current values which is used for rank prediction. |
| User defined | Prediction Formula | Protected | - | This is the Prediction formula which is currently selected to compute automobile rank. |

***3.2.3.2.1 predict()****:*

Purpose: This function is used to get the prediction for selected automobile with current parameters and prediction formula.

Input: This function uses global parameter list and prediction formula to predict rank

Output: This function returns the predicted rank of the selected automobile as output

Exceptions: The global parameter list or prediction formula may not be present initially, if the administrator has not set it. Hence an exception has to be raised in this case.

Pseudo code:

predict(){

for selected automobile, access current parameter values from parameter list

result=prediction formula using parameters

return result

}

***3.2.3.2.2 sortOutput()****:*

Purpose: This function is used to sort the automobile ranks based on some sorting order(ascending or descending).

Input: This function uses global output list as input and sorts it.

Output: This function returns the sorted output list

Exceptions: If there is no output previously generated, output list will be empty and hence an exception has to be raised as there are no values to sort.

Pseudo code:

sortOutput(){

outlist=access global output previously generated.

apply sorting algorithm on outlist(ascending by default)

return outlist

}

***3.2.3.2.3 applyfilters()****:*

Purpose: This function is used to apply different filters on the output list.

Input: This function uses global output list as input and applies filters on it based on user selection

Output: This function returns the new output list with filters

Exceptions: If there is no output previously generated, output list will be empty and hence an exception has to be raised.

Pseudo code:

applyfilters(){

outlist=access global output previously generated.

newlist=outlist after applying user selected filters(provided as global filter variables).

return newlist

}

***3.2.3.2.4 displayranks()****:*

Purpose: This function is used to display ranks of all the selected vehicles.

Input: This function does not take any input parameters

Output: This function returns a list of outputs which are the ranks of the selected vehicles.

Exceptions: If no vehicles are selected, then an exception has to be raised.

Pseudo code:

display ranks(){

selected=get the selected vehicles from database

output=[]

for i in selected:

rank=predict() for i

output.append(rank)

return output

}

***3.2.3.2.5 addParam(parameter, value)****:*

Purpose: This function is used to add a parameter to the parameter list.

Input: This function uses the new parameter name and it’s value as inputs.

Parameters: This function takes the new parameter and a value as parameters

Output: This function adds the parameter to the parameter list.

Exceptions: If the parameter provided is invalid or has a value which is impossible, it cannot be added to the parameter list and an exception has to be raised.

Pseudo code:

addParam(parameter,value){

paramlist=retrieve global parameter list

if(parameter is valid && value is valid)

{

paramlist.append(parameter,value)

update parameter list on database

}

else

throw exception

}

***3.2.3.2.6 delParam(parameter)****:*

Purpose: This function is used to delete a parameter from the parameter list.

Input: This function uses the existing parameter name.

Parameters: This function takes the parameter name which is a string value as function parameter.

Output: This function removes the parameter from the parameter list.

Exceptions: If the parameter name provided does not exist in the parameter list, an exception has to be raised.

Pseudo code:

delParam(parameter){

paramlist=retrieve global parameter list

if(parameter in parameter list)

{

paramlist.remove(parameter)

update parameter list in database

}

else

throw exception

}

***3.2.3.2.7 alterParam(parameter, value)****:*

Purpose: This function is used to alter an existing parameter from the parameter list.

Input: This function uses the parameter name and it’s new value as inputs.

Parameters: This function takes the new parameter and a value as parameters

Output: This function updates the parameter value in the parameter list.

Exceptions: If the parameter provided is not present in the parameter list or the new value provided is of wrong type, an exception has to be raised.

Pseudo code:

alterParam(parameter,value){

paramlist=retrieve global parameter list

if(parameter in paramlist && value is valid)

{

paramlist.update(parameter,value)

}

else

throw exception

}

**3.2.4 Parameter**

**3.2.4.1 Class Description:**

This is used to represent the class of parameters used in rank prediction. Each parameter has a name and a value associated with it. This value can be either retrieved(using getval() operation) or changed(using changeval() operation). The Parameter class exists as a part of the Predictor tool class and cannot exist independently, hence the composition.

**3.2.4.2 Data Members:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Data Name** | **Access Modifiers** | **Initial Value** | **Description** |
| String | Name | Protected | - | Each parameter has a unique name which is present in the parameter list and used to identify and alter the parameter. |
| Number | Value | Protected | - | This is the current value associated with that parameter which is used when the parameter is used in the prediction formula. |

***3.2.4.2.1 getVal(name):***

Purpose: This function is used to get the current value of the specified parameter.

Input: This function uses the parameter name as input.

Parameters: This function takes a string value(name) as parameter.

Output: This function returns the current value of the specified parameter.

Exceptions: If the parameter specified does not exist in parameter list, an exception has to be raised

Pseudo code:

getVal(name){

value=retrieve parameter with parameter name=name from database

if(value)

return value

else

raise Exception

}

**3.2.5 Prediction Formula**

**3.2.5.1 Class Description**

This is used to represent the weighted formula used for automobile rank prediction. Different types of formula (such as linear, exponential or priority based) are present which can be used according to the type of vehicle or type of rank prediction required. The operation changeFormula() can be used to switch between the formulae. This class also exists as a composition of Predictor tool class

**3.2.5.2 Data Members**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Data Name** | **Access Modifiers** | **Initial Value** | **Description** |
| String | Type | Protected | - | This denotes the type of the formula used for prediction, such as linear, exponential, logarithmic and so on. |

***3.2.5.2.1 changeFormula(type)***

Purpose: This function is used to change the type of the formula to given type.

Input: This function takes the desired type as input.

Parameters: This function takes a string value(type) as parameter.

Output: This function changes the formula to another formula of type “type”

Exceptions: If there is no formula for given type, an exception has to be raised

Pseudo code:

changeFormula(type){

globalformula=choose formula from database of given type

update current formula

}

## 3.3 Database Module

### 3.3.1 Database Admin:

#### 3.3.1.1 Class Description:

This class is used to represent the database admin. The database admin is responsible for collecting data from various sources and feeding it into the database. The database admin is responsible for modifying values in the table if the car test values change over time. The Database admin is also responsible for removing data of cars from the table that are no more in production. If a new company starts manufacturing cars, the database admin has to create a new table to store the car information of the new company.

3.3.1.2 Data Members:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | Data Name | Access Modifiers | Initial Value | Description |
| String | Username | Protected | - | Each Database admin has a unique username. |
| String | Password | Private | - | Each database admin has a unique password that is associated with the username. |

*3.3.1.2.1* add\_company()

Purpose: This method is to add a new company to the database.

Input: This takes name of the company as input.

Exceptions: The admin may a name of the company that already exists so for this the database admin has to manage it.

Pseudo-code: CREATE TABLE <company\_name> (column\_1 datatype, column\_2 datatype, column\_3 datatype);

*3.3.1.2.2* delete\_company()

Purpose: This method is to delete an old company that no more produces cars from the database.

Input: This takes name of the company as input.

Exceptions: The admin may a name of the company that does not exist.

Pseudo-code: DROP TABLE <company\_name>

*3.3.1.2.3* add()

Purpose: This method is to add to the database.

Input: This company takes name of the company and performance test values as input.

Exceptions: The admin may give a performance test value that is out of bound.

Pseudo-code: INSERT INTO <company\_name> VALUES (value1, value2,….valueN);

*3.3.1.2.4* delete()

Purpose: This method is to delete a row of a table in the database.

Input: This takes name of the company and key value of the row to delete.

Exceptions: The admin may give wrong key.

Pseudo-code: DELETE FROM <company\_name> WHERE <condition>

*3.3.1.2.5* modify()

Purpose: This method is to modify a row of a table in the database.

Input: This takes name of the company and key value of the row to modify, it also takes in the new values that have to replace existing data as input.

Exceptions: The admin may an out of bound value.

Pseudo-code: UPDATE <company\_name> SET col1=val1, col2=val2… [Where condition];

### 3.3.2 Vehicle Database

#### 3.3.2.1 Class Description:

This is the database into which the database admin feeds data.

#### 3.3.2.2 Data Members:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | Data Name | AccessModifiers | Initial Value | Description |
| list | Company Names list | Protected | - | Each table in this database represents vehicle information of all cars of a particular car manufacturing company. |

*3.3.2.2.1* create\_table()

Purpose: On receiving request from database admin the database has to create tables.

Input: Company name.

Exceptions: Name of the company should not repeat.

*3.3.2.2.2* delete\_table()

Purpose: delete a company table from table.

Input: Company name.

Exceptions: Company name should exist in the database.

### 3.3.3 Vehicle table

#### 3.3.3.1 Class Description:

This class represents the table of each company

#### 3.3.3.2 Data Members:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | Data Name | Access Modifiers | Initial Value | Description |
| String | Company name | Protected | - | Each company has its own table. |
| float | Test values | Protected | - | These are the test values that are given by the database admin. |

### 3.3.4 User

**3.3.4.1 Class Description:**

This is the user who wants data from the database to perform comparison and generate a rank.

This class has attributes username and password. The operation that this class supports is api\_call(). This function provides data based on the api\_call() from the user.

**3.3.4.2 Data Members:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Type** | **Data Name** | **Access Modifiers** | **Initial Value** | **Description** |
| username | Parameter list | Protected | - | Username of user which is unique. |
| password | Prediction Formula | private | - | Password of the user |

***3.3.4.2.1*** api\_call(api\_url):

Purpose: The purpose of this method is to provide data to the user who wants to predict the rank based on the url.

Input: The url to retrieve the data from.

Output: The data retrieved from the database.

Parameter: api\_url is the parameter.

Exceptions: User provides invalid url.

Pseudo code:

api\_call(api\_url)

{

Q= Generate query based on api\_url

Data=retrieve data from database based on Q

Return Data

}

# **Traceability Matrix**

|  |  |  |
| --- | --- | --- |
| **CRS Reference Section No. and Name** | **DESIGN / HLD Reference Section No. and Name** | **LLD Reference Section No. Name** |
|  | predict(), sortOutput() | ***3.2.3.2.1,3.2.3.2.2*** |
|  | changeFormula() | ***3.2.5.2.1*** |
|  | login() | ***3.1.4.1.1*** |
|  | authorizeUser() | ***3.1.5.2.2*** |
|  | signUp() | ***3.1.3.1.1*** |
|  | checkPassword() | **3.2.1.2.1** |
|  | blockUser(), authorizeUser(), confirmUserChanges() | ***3.1.1.1.1, 3.1.5.2.2, 3.1.5.2.3*** |
|  |  |  |
|  |  |  |
|  |  |  |